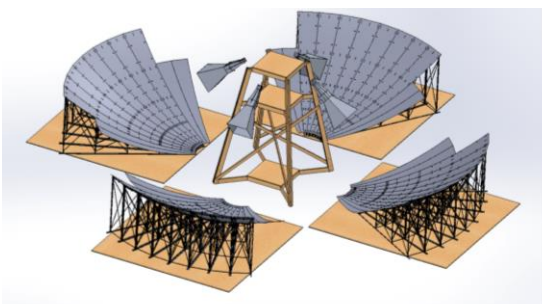
BMX Radio Spectrometer

Hindy Drillick

7/6/17

BMX Telescope

- Purpose: mapping the 21-cm emission line of hydrogen
- A 21 cm wavelength has frequency of 1.42Ghz (z=0), 1.11Ghz (z=.3)
- So we are looking at radio frequencies in range 1.1 – 1.65 Ghz



- 4 dishes and horns
- 2 channels per horn- for X/Y polarization
- Currently only have 2 channels

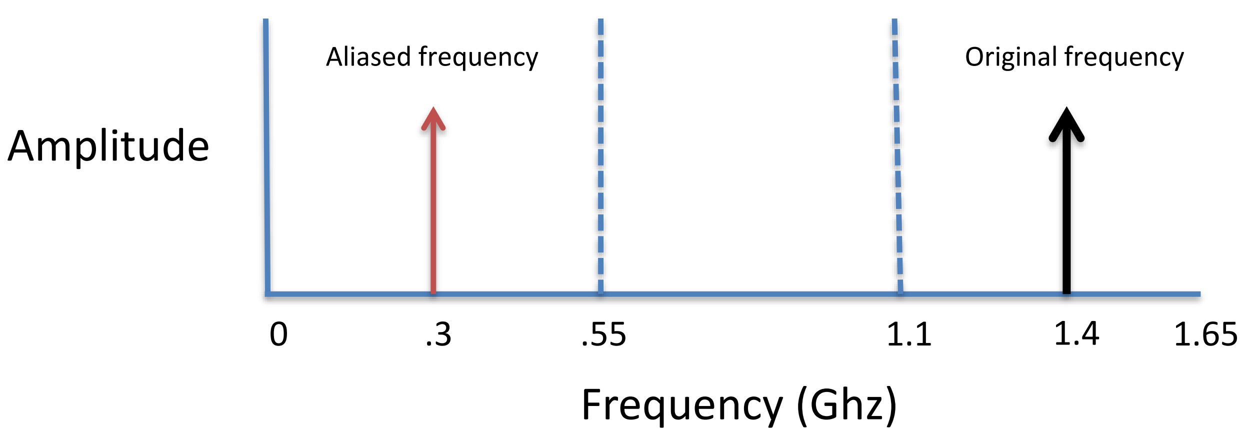
June 28, 2017 – Assembly of BMX telescope dish, Brookhaven National Lab

Data Acquisition

- 2 channels attached to an 8 bit ADC (analog to digital converter/digitizer)
- Signal sampling rate: 1.1 Ghz
- Data is processed in packets of 2^{27} bytes (per channel)
- $2^{27}/1.1e9 = 122.016$ ms of data per packet

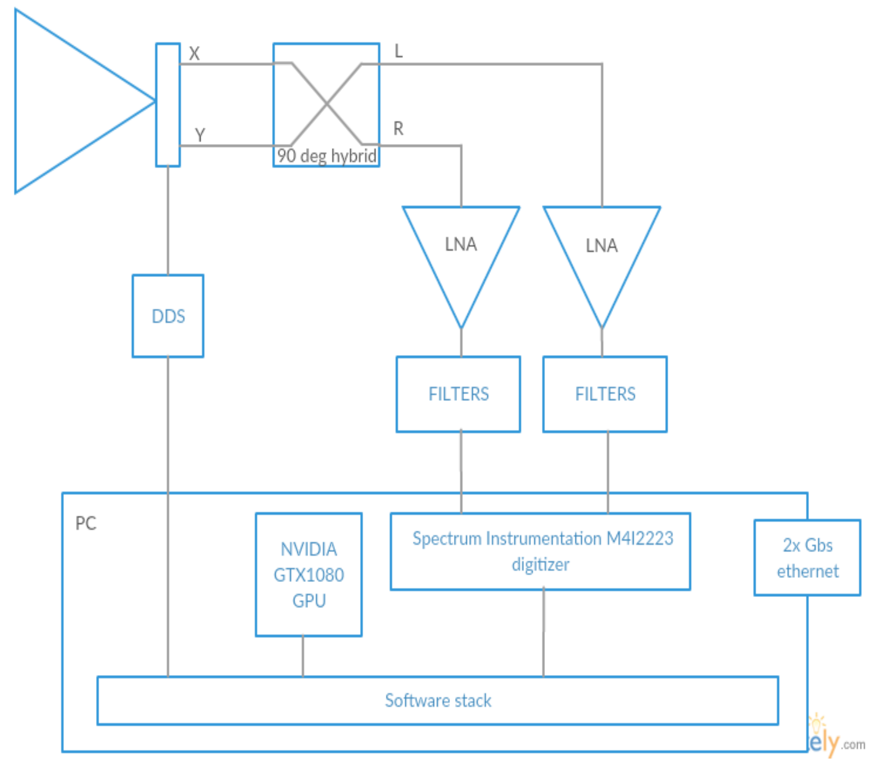
<u>Aliasing and Filtering</u>

- We accurately measure signals up until half the sampling frequency (.55 GHz)
- But we need frequencies in the range 1.1 – 1.65 Ghz
- Images of signals > .55Ghz will get projected back onto the range 0 - .55Ghz,
- A filter blocks all signals outside 1.1 – 1.65 Ghz, so that the aliasing of this range won't mix with other frequencies.



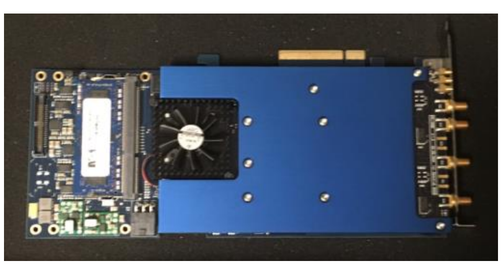
Data Pipeline

1. Data acquisition
2. Copy data from CPU (host) memory to GPU (device)
3. RFI (radio frequency interference) rejection
4. Perform Fast Fourier Transform
5. Compute power and cross power spectra
6. Copy reduced data back to host
7. Write spectra to file



All of computations happen on a GPU (Graphics Processing Unit)

- Hardware: Nvidia GeForce GTX 1080
- Software: CUDA – an API for parallel computing on a GPU



My contribution

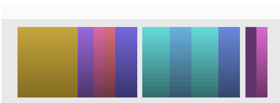
Before I came here, the spectrometer was operational. My task was to

1. Streamize GPU operations to enable concurrent processing and memory transfer. This allows more GPU cycles to be available for computation.
2. Implement a simple RFI rejection mechanism.

CUDA Streams and Kernels

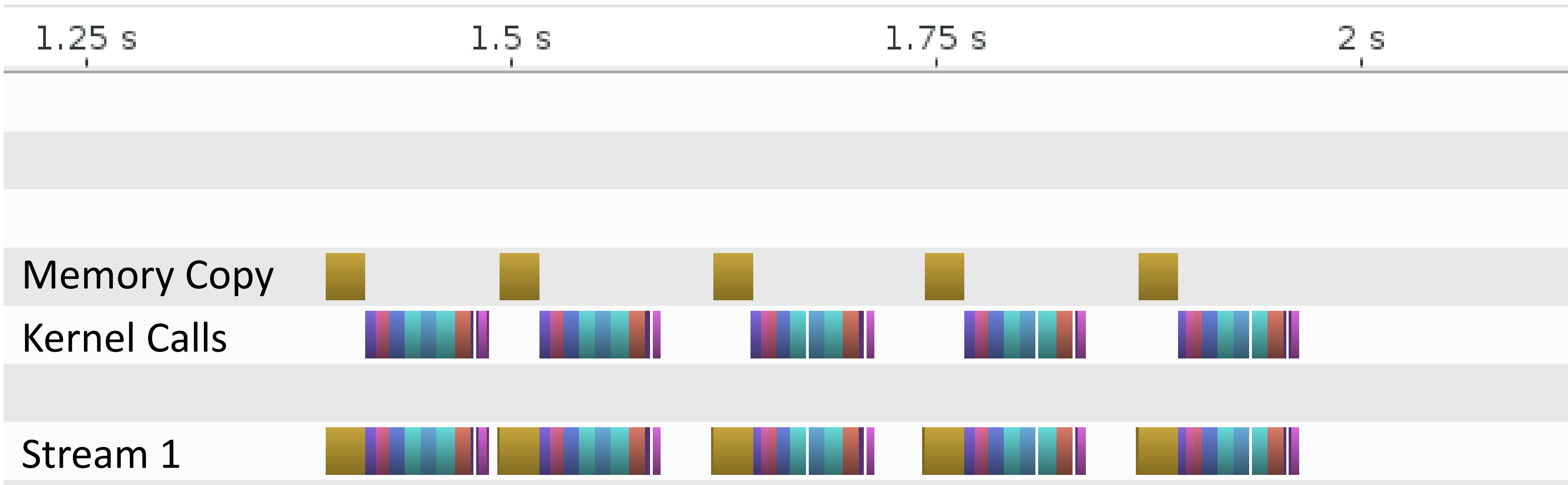
- A kernel is a function that runs on the GPU.
- A stream is a sequence of CUDA operations that are executed in serial order on the GPU
- Operations in different streams can run concurrently
- The three main CUDA operations that we use are
    1. Data copy from CPU to GPU
    2. Kernel calls (e.g. FFTs, compute power spectra)
    3. Data copy from GPU to CPU

One Stream, 2 Channels



One data packet

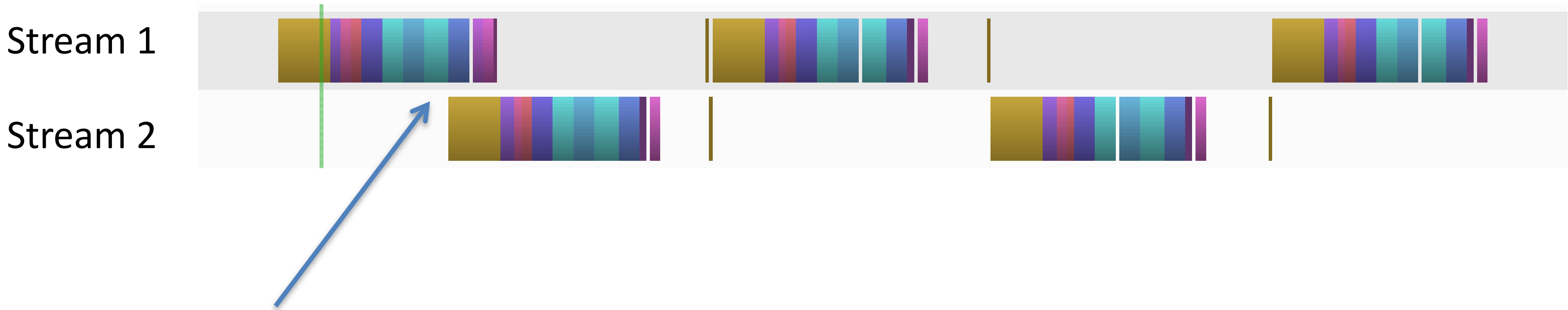
Nvidia Visual Profiler -NVVP



Issues:
Packets are handled serially. If we need more than 122 ms to analyze the data, then the digitizer/ADC buffer will eventually overflow, as packets pile up.

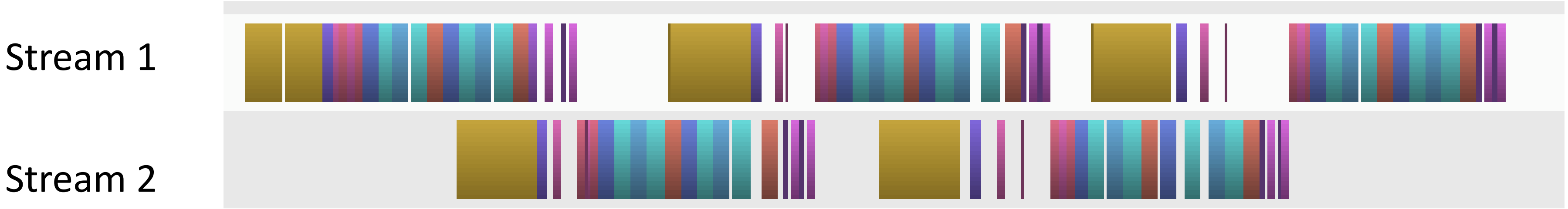
Multiple Streams, 2 Channels

- Digitizer memory needs to be pinned (stays on RAM)



- Now have concurrent data transfers and kernel execution
- Can handle 2 or 3 streams
- With 4 streams, runs out of memory, because needs to allocate separate 1.02 Gb memory buffers for each stream.

Multiple Streams, 4 Channels



- Doubling all operations, to mimic 4 channels
- Concurrent data transfers and kernel execution

Issues:

- Kernels not executing concurrently with one another

Why?

- Kernels run on multiple threads, which are grouped into blocks
- GeForce GTX 1080 contains 20 multiprocessors, and each processor handles up to 32 blocks at a time. Our kernels use more than 640 thread blocks, so there are no free blocks until the kernel is finished.

Radio Frequency Interference (RFI)

- Radio signals with terrestrial origins e.g TV, wifi, cell phone signals that corrupt our astronomical data

Detection:

- Divide data packet into 2^7 chunks of 2^{20} numbers each and calculate variance (σ^2) for each chunk
- Calculate the mean and standard deviation (rms) of σ^2 across all the 2^7chunks
- Flag chunk as outlier if σ^2_i - σ^2_{mean} > N*rms, where N is some integer that we choose
- Other statistics such as mean, and absolute max can be used instead of variance

Radio Frequency Interference (RFI)

- Radio signals with terrestrial origins e.g TV, wifi, cell phone signals that corrupt our astronomical data

Detection:

- Divide data packet into 2^7 chunks of 2^{20} numbers each and calculate variance (σ^2) for each chunk
- Calculate the mean and standard deviation (rms) of σ^2 across all the 2^7chunks
- Flag chunk as outlier if σ^2_i - σ^2_{mean}> N*rms, where N is some integer that we choose
- Other statistics such as mean, and absolute max can be used instead of variance

Channel 1	0	1	0	1	0	0	0	0
Channel 2	0	0	0	1	0	1	0	0

RFI Rejection

1. Zero out the values in the rejected chunks before performing FFT
2. Write chunk out to file for further examination (1.04 Mb) in case astronomically significant
3. Adjust power spectra based on number of chunks that were nulled out:

Channel 1: $P(f) = P(f) * \frac{nChunks}{(nChunks - nNulledChannel1)}$

Channel 2: $P(f) = P(f) * \frac{nChunks}{(nChunks - nNulledChannel2)}$

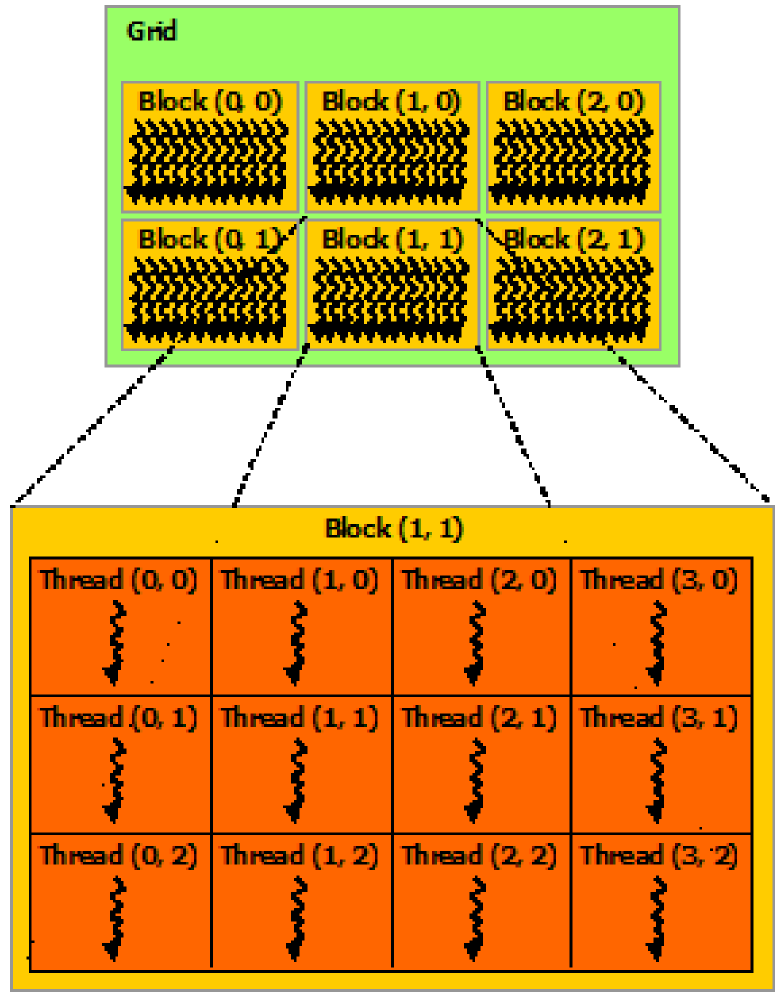
Channel 1 x Channel 2:

$$P(f) = P(f) * \frac{nChunks}{(nChunks - nNulledChannel1\_OR\_Channel2)}$$

Channel 1	0	1	0	1	0	0	0	0
Channel 2	0	0	0	1	0	1	0	0
Channel 1 OR Channel 2	0	1	0	1	0	1	0	0

CUDA Kernels

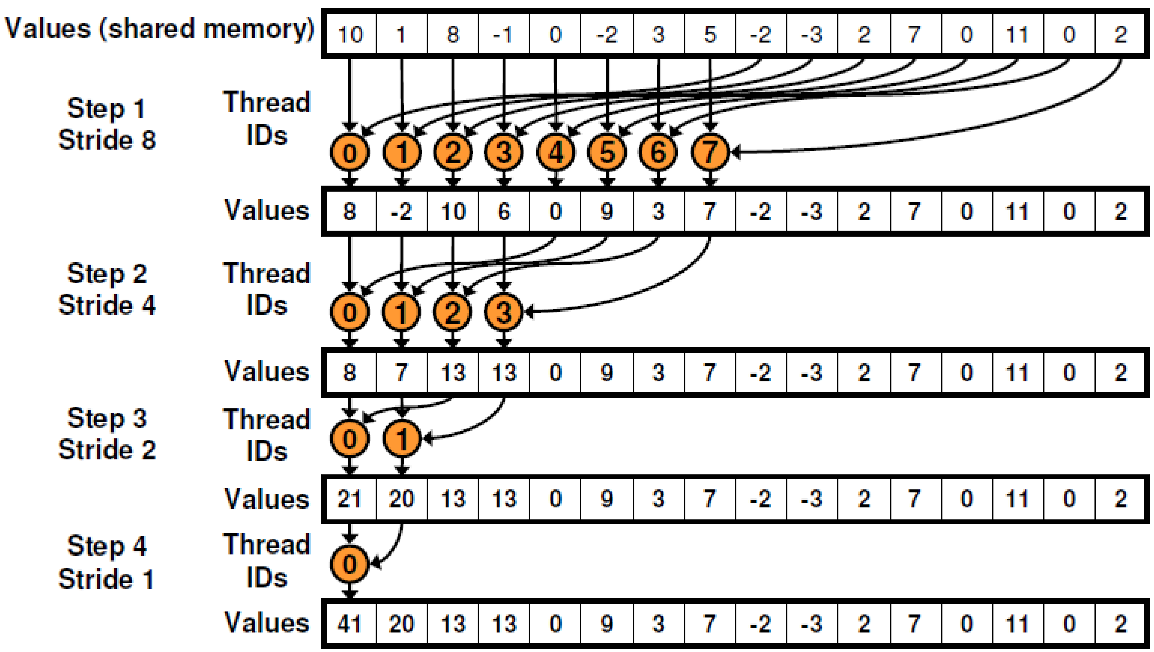
- A CUDA kernel is a function that is launched in parallel on multiple thread blocks
- Memory can be shared by threads in the same block
- Choose number of blocks, and number of threads per block
- Limit of 1024 threads per block, and 2^31 blocks per kernel call
- When executing, the kernel knows which thread and block it is running on



Nvidia Programming Guide

Kernel to calculate sum

- To calculate the sum of many numbers on a GPU, we use a parallel reduction algorithm
- Need number of elements to be power of 2
- Each thread loads an element into its block's shared memory



- Calculates separate sum per block – need recursive kernel calls to then sum the blocks
- Simple variations of this algorithm give mean, max and sum of squares
- O(log n)

Going forward

- Optimize computations further if possible, so can handle more computation per packet
- Determine best parameters to detect RFI